



# week 1

## 프로젝트 개발 순서

- 스프링 프로젝트 생성 
- 스프링 부트로 웹 서버 실행 
- 회원 도메인 개발
- 웹 MVC 개발
- DB 연동 - JDBC, JPA, 스프링 데이터 JPA
- 테스트 케이스 작성

## 사용 프레임워크 / 라이브러리

### Spring boot

- spring 프레임워크를 보다 쉽게 사용할 수 있도록 만든 프레임워크
- 개발자가 설정 파일을 작성할 필요 없이, 프로젝트의 설정과 라이브러리 의존성을 자동으로 처리해주는 기능을 제공

### Gradle

- groovy를 기반으로 한 빌드 도구
- 종속성 관리, 컴파일, 패키징, 테스트, 배포 등 여러 작업을 수행

### Thymeleaf

- java XML / XHTML / HTML5 용 템플릿 엔진. 완전한 스프링 프레임워크 통합 제공
- html에 스크립트를 추가해 동적으로 페이지를 구성할 수 있게 해주는 엔진

### JPA

- Java EE(enterprise edition)을 관리하는 회사가 바뀌면서 Jakarta EE로 이름이 바뀌고, JPA 또한 Jakarta Persistence 로 변경  
API 패키지명은

`javax.*` 에서 `jakarta.*` 로 변경

- ORM(Object-Relational Mapping) 기술 표준으로 사용되는 인터페이스의 모음
- 인터페이스기 때문에 Hibernate, OpenJPA 등이 JPA를 구현

<https://github.com/jakartaee/persistence/tree/master/api/src/main/java/jakarta/persistence>

- ORM이란? 객체와 관계형 데이터베이스의 데이터를 자동으로 연결시켜주는 것

### Hibernate

- 자바 ORM 프레임워크, JPA의 구현체 중 하나
- 메서드를 호출하면 jdbc 드라이버를 통해 내부적으로 sql 쿼리를 수행

## Spring initializr

<https://start.spring.io/>

스프링 부트 기반으로 프로젝트를 생성할 때 사용하는 사이트

빌드 도구, 언어, 버전 등을 설정하면 초기 설정이 되어있는 프로젝트 압축 파일을 다운받을 수 있음

최근에는 maven 보다 gradle을 더 많이 사용

SNAPSHOT, M1 버전은 정식 릴리즈가 아니기 때문에 사용 자제

## JAR, WAR

자바 앱을 패키징하는 방식

- JAR(Java ARchive)은 class 파일과 그 외에 필요한 다른 파일들을 패키징하는 파일 포맷
- WAR(Web application ARchive)은 Servlet / Jsp 컨테이너에 배치할 수 있는, 웹 앱을 구성할 때의 필요한 요소들의 압축 파일로, 웹서버 혹은 웹 컨테이너가 필요함

## groupId, artifactId

groupId: 프로젝트의 그룹 혹은 회사

artifact: 프로젝트의 이름

## 프로젝트 구조

### .idea

인텔리제이가 사용하는 설정 파일

### src

- 소스 코드를 저장하는 디렉토리
- main과 test로 나뉘져 있고,  
main/java/<group>.<artifact> 안에 프로젝트 소스코드를 저장  
test/java/<group>.<artifact> 안에 테스트 코드를 저장
- resources는 자바 파일을 제외한 설정 파일, html등을 저장

## build.gradle

spring initializr에서 설정한 스프링 부트, 자바 버전, 그룹,  
사용할 라이브러리, 라이브러리 저장소 등이 적혀있음

## 라이브러리

현재 프로젝트에 설치된 라이브러리들을 보면 그 수가 spring initializr에서 설정한 것보다 훨씬 많은 것을 볼 수 있는데, spring-web과 thymeleaf가 작동하기 위해 필요한 라이브러리들을 모두 다운받아오기 때문

### spring-boot-starter-tomcat

스프링 부트는 아파치 톰캣 웹 서버가 내장되어있어 따로 설치할 필요가 없음

서버를 띄우고 무언가를 출력할 때 stdout이 아닌 로그로 출력을 해줘야 하는데, 이를 위해 spring-boot-starter-logging 라이브러리가 사용되고, 종속되는 라이브러리로는

- slf4j (인터페이스)
- logback (구현체)
- log4j (구현체)

가 있음

테스트를 위한 spring-boot-starter-test는

- junit (자바 진영에서 가장 많이 사용되는 테스트용 라이브러리)
- assertj, mockito (테스트를 편리하게 할 수 있도록 도와주는 라이브러리)

## View 환경설정

`src/main/resources/static/index.html` 을 생성하면 웰컴 페이지 기능을 제공

스프링은 index.html 이라는 파일을 기본 웰컴 페이지로 사용

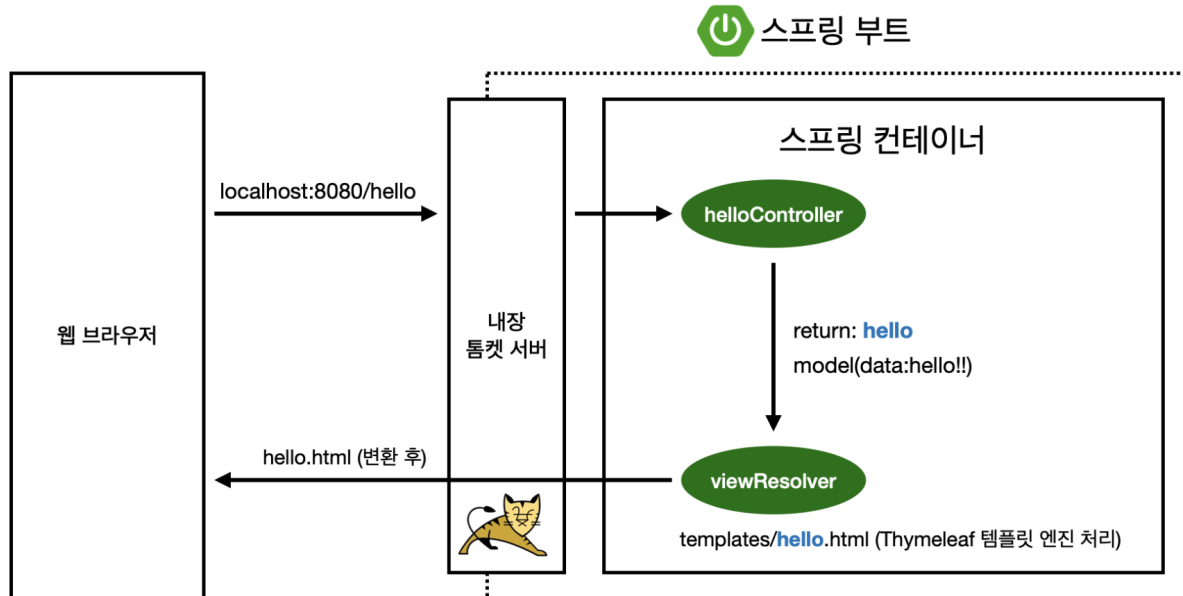
스프링이 지원하는 템플릿 엔진

- FreeMarker
- Groovy
- Thymeleaf
- Mustache

컨트롤러에서 문자를 반환하면 뷰 리졸버가 화면을 찾아서 처리

- 스프링 부트 템플릿 엔진 기본 viewName 매핑
- `resources:templates/` + `{viewname}` + `.html`

## 동작 환경 그림



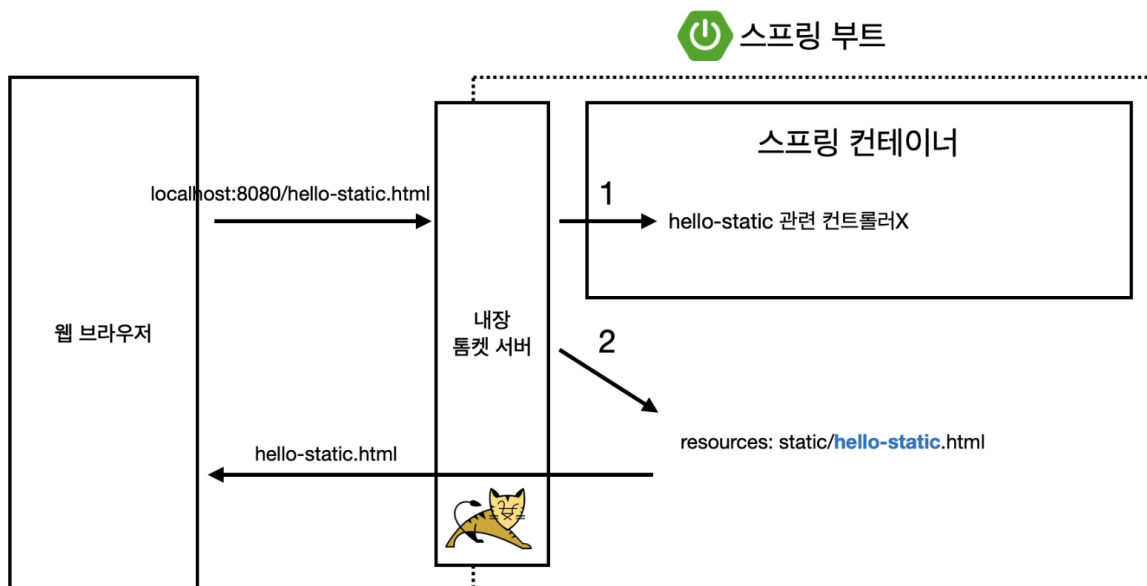
## 스프링 웹 개발 기초

### 정적 콘텐츠

서버의 추가적인 처리 없이 클라이언트에게 넘겨주는 파일

기본적으로 `/static`, `/public`, `/resources`, `/META-INF/resources` 디렉토리에서 파일을 찾아 넘김  
위 디렉토리들 중 하나에 파일을 생성하고,

`address:port/example.html` 으로 접속하면 파일을 받을 수 있음



## MVC와 템플릿 엔진

서버의 추가적인 처리로 파일의 내용이 변경되어 클라이언트에게 넘겨주는 방식

### MVC 패턴

소프트웨어 구조를 위한 디자인 패턴 중 하나. 주로 UI를 가진 앱에 사용되며, model, view, controller 세 부분으로 나누어져 있다.

Model

- 앱의 데이터를 나타냄

View

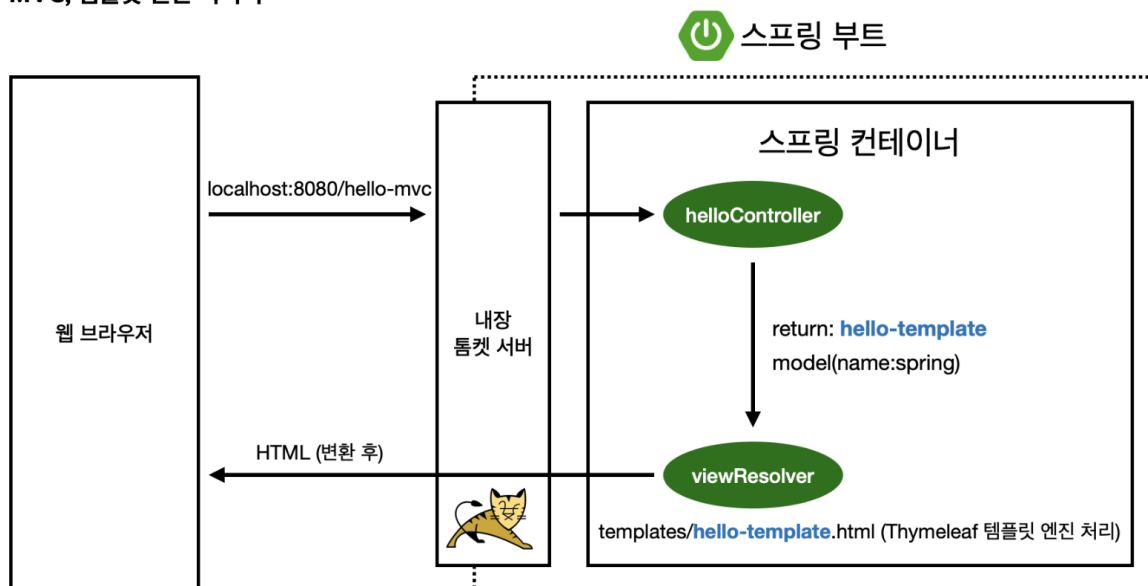
- 사용자와 상호작용할 수 있는 인터페이스

Controller

- Model과 View의 상호동작을 관리

과거 모델 1 방식은 View와 Controller의 구분이 없었지만 지금은 둘을 구분해서 개발  
View는 화면과 관련된 일만 처리, Controller는 그 외의 비즈니스 로직을 담당

### MVC, 템플릿 엔진 이미지



### query parameter

`example.com/?param=example&param2=example`

```
// required는 기본적으로 true로 세팅되어있음
// 쿼리 파라미터에 exam이 없으면 서버는 400 Bad request를 뱉음
@GetMapping("example")
public String example(@RequestParam("exam") String exam, Model model) {
    return "example";
}
```

```
// required를 false로 지정한다면, 200 Ok를 뱉지만, 파라미터인 exam에는 NULL이 들어
@GetMapping("example2")
public String example(@RequestParam("exam", required = false) String exam) {
    return "example2";
}
```

## API

데이터를 xml, json 구조의 포맷으로 넘겨주는 방식

view가 필요 없는 상황에 데이터만 보내주기 위해 사용

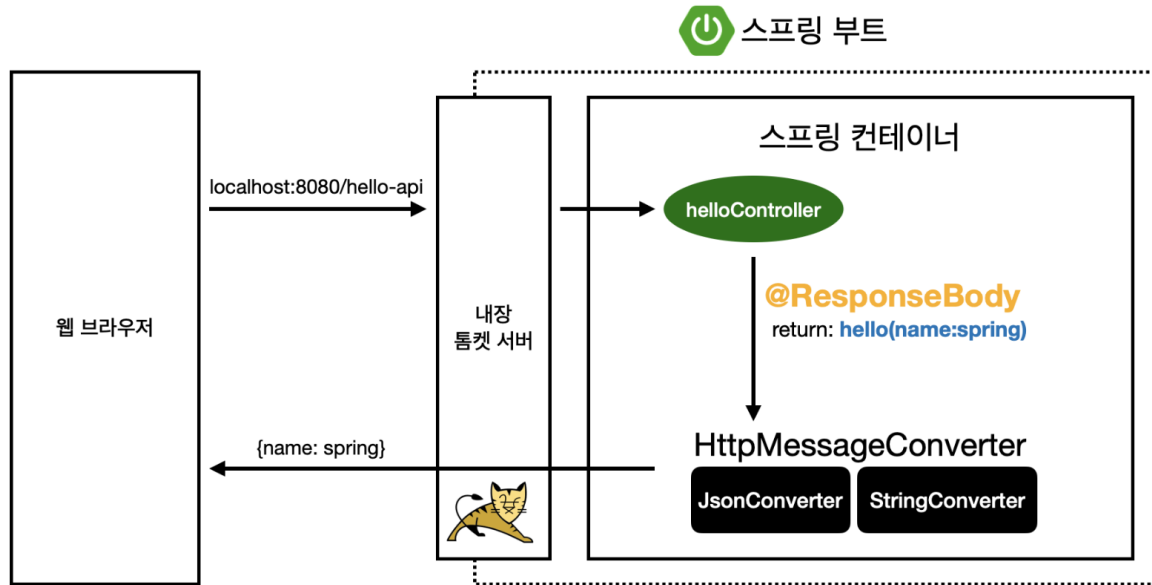
```
@GetMapping("example")
@ResponseBody // 객체나 문자열을 HTTP 응답 body의 내용으로 매핑하는 역할
// ViewResolver 대신에 HttpMessageConverter가 동작
public String helloString(@RequestParam("name") String name) {
    // 문자열을 리턴하면 response body에는 텍스트가 담겨 전송
    // StringHttpMessageConverter가 사용됨
    return "hello " + name;
}

@GetMapping("hello-api")
@ResponseBody
public Hello helloApi(@RequestParam(value = "name", required = false) String name) {
    // 객체를 리턴하면 기본 설정으로는 json으로 변환해 response body에 담아 전송
    // MappingJackson2HttpConverter가 사용됨
    Hello hello = new Hello();
    hello.setName(name);
    return hello;
}

static class Hello {
    private String name;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```



## HttpMessageConverter

인터페이스이며, `canRead()`, `canWrite()`, `read()`, `write()` 메서드가 존재

`canRead()`, `canWrite()` 는 메세지 컨버터가 해당 클래스, 미디어 타입을 지원하는지 확인

`read()`, `write()` 는 메세지 컨버터를 통해서 메시지를 읽거나 씀

문자열의 경우에는 `StringHttpMessageConverter` 를 사용

객체나 HashMap은 `MappingJackson2HttpConverter` 를 사용

이외에 케이스에 대해서는 해당하는 컨버터들이 존재하고, 올바른 컨버터를 찾지 못한다면 예외가 발생